



Function Anti-Patterns

Alberto Manuel Simões
 ambs@di.uminho.pt

In the last couple of issues of *The Perl Review*, I've written about how *not* to do things. This time, I would like to talk about methods, functions, subroutines or whatever you like to call them. I present a few tips to make functions easier to use, less buggy, and more readable.

What's the difference?

First, I should explain the basic difference between a method and a function. A method is the object-oriented version of a function: it is a function called on an object. In Perl there isn't a syntactic difference between the two. When I call a function, Perl passes just the arguments I put in the parentheses.

```
copy_to_dir( $file, $dir );
```

On the other hand, if I use the arrow notation with an object, I'm calling a method. In this example, I use the method named `finish()`, which is just a Perl function that gets the object as its first argument automatically.

```
$database->finish;
```

Perl converts this to this function call (along with some other magic for inheritance):

```
DBI::finish( $database );
```

Naming Conventions

As with any variable, name functions so other programmers can easily figure out what they do. A good name can, in many cases, substitute for documentation (but you should not skip the documentation). Choose a simple name that reads well in English.

For instance, if I am writing a method to check for a property that returns a boolean value, I give it a name that reads well when I use it in an `if` statement. A bad example doesn't read like a sentence. What does the return value of `getActiveState()` mean? What decision am I trying to make based on it?

```
if( $obj->getActiveState() ) {
    #...
}
```

Instead, I try to make a sentence out of it. In this example, I'm telling other programmers that the block should only run if the object is active (whatever that means for that hypothetical object).

```
if( $obj->isActive() ) {
    #...
}
```

Also, I choose a coherent and consistent naming style for all of my code. Perl allows me to make quite long names (how much memory do you have?), so I don't have to remove all of the vowels:

```
smfnctn( @args )
```

I program so that the next person after me can figure out what I'm doing. It's easier to say it so the next guy doesn't have to guess. This can be especially hard if the cryptic name comes from a language the reader doesn't know, or doesn't know well. It's not a puzzle after all (unless you are in an obfuscated Perl contest, where anti-patterns turn into good patterns). Use full words and common names.

```
some_function( @args );
```

Several naming schemes exist, and I should be consistent. I can use CamelCase, where I cram all of the words together without anything but capitalization to separate them, or the more Perly `some_function` naming schemes for functions where I separate words with underscores. If I'm working with someone else's code, I should follow their naming scheme (at least until I can convince them to do it right!).

Argument Management

Perl doesn't handle function signatures with argument names, so I have to do that myself in the code. My code needs to be clear, and just as function names help to document the code, so do good variable names. In this example, what are `$p` and `$v`? Maybe